Assignment for week 8: Sorting

This assignment is due at noon on Friday, 12 March.

In what follows, "develop a class" and "develop a method" mean what they meant in assignment 6. You are *not* required to use templates for this assignment.

The purpose of this homework is to implement three different sorting algorithms, each working with two different kinds of underlying data structures (list or vector). The algorithms will then be tested on several different input data sets, using two different ways of comparing the elements of the structure.

For this assignment, you are given a file named `Week8.java` that contains some of the code from `Lab7.java` and some additional classes that you can use to construct examples and test data for this assignment.

1. Develop a class `City` that contains information about a US city's name, state, and geographical location given in latitude and longitude. The state will be given as a standard two-letter abbreviation. The latitude and longitude will be accurate to four or five decimal places.

2. Develop two different classes that implement the `Comparator` interface. Instances of the first class shall compare two cities according to the lexicographic order induced by the natural orderings on the strings that give the cities' state abbreviation and name, considering the state abbreviation to be more significant than the name. Instances of the second class shall compare two cities according to their latitude, with the more southern city being considered less than a more northern city; if two cities have equal latitudes, then this second `Comparator` shall regard them as equal.

3. Develop a `VectorRanger` class that implements the `IRange` interface. The constructor for this class shall take a `Vector` as its sole argument. An instance of this class shall generate the elements of that `Vector`, in order.

4. The `SortAlgorithm` interface is defined by

   ```
   interface SortAlgorithm {
   ```

```
// Stores the objects generated by the given IRange into
// some local data structure, but does not sort them.
//
// This method must be called before any of the others.
// This method may be called more than once, in which
// case the objects that were generated for a previous
// call are discarded.

void initializeDataSet(IRange ir);

// Sorts the objects into some internal data structure without
// changing the local data structure that holds the unsorted
// objects.
//
// (Thus a destructive sort algorithm must first perform a
// shallow copy of the objects to be sorted into the data
// structure that will be destroyed by the sort() method.)
//
// Repeated calls to this method should repeat all the steps
// of the first call, because this method will be timed to
// measure the performance of different sort algorithms.
//
// This method shall not be called until after this object's
// initializeDataSet(IRange) method has been called.  This
// method shall be called before this object's
// isSorted(Comparator) method is called.

void sort(Comparator cmp);

// Returns true if the sorted objects are in order according
// to the given Comparator, which is not necessarily the one
// that was used to perform the sort.
//
// This method shall not be called until after this object's
// sort(Comparator) has been called.

isSorted(Comparator cmp);

// Returns an IRange that generates the sorted objects, in
// sorted order.
```

```
    //
    // This method shall not be called until after this object's
    // sort(Comparator) has been called.
    // If this method is called more than once, it shall return
    // a freshly created IRange for each call.

    IRange result();
}
```

Develop six classes

- `ListInsertionSort`,
- `VectorInsertionSort`,
- `ListMergeSort`,
- `VectorMergeSort`,
- `ListQuickSort`,
- `VectorQuickSort`

that implement the `SortAlgorithm` interface using the named algorithm (insertion sort, merge sort, or quicksort) on the named data structure (AList or Vector). The constructors for these classes shall take no arguments.

5. For each class, develop the four methods listed in the `SortAlgorithm` interface.

6. Write tests for the six implementations of `SortAlgorithm`, using both `Comparator` classes. (The `randomCities(int)` method in class `CityData` might be helpful here.) Change the `main` method so it runs your tests.

7. Submit your program in the usual way.